2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Optimizing Random Test Constraints Using Machine Learning Algorithms

Stan Sokorac

stan.sokorac@arm.com

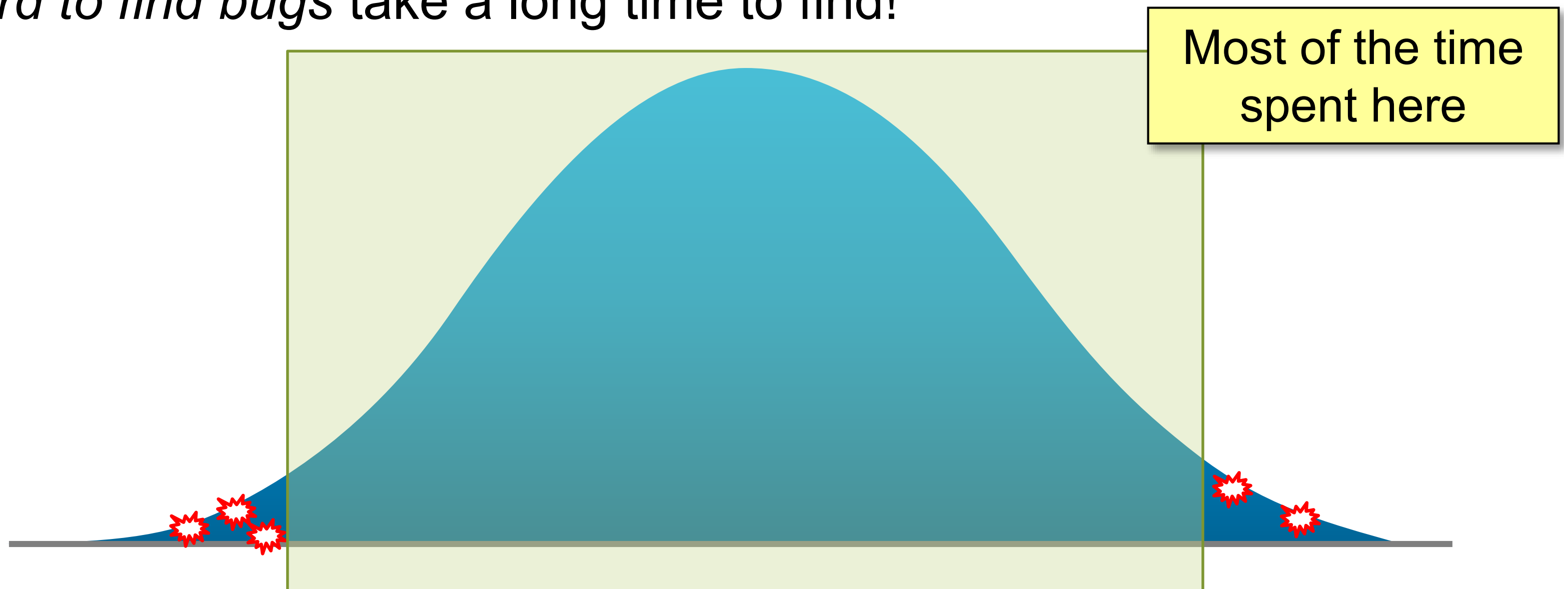
ARM

Background

- Modern designs are extremely complex
  - Impossible to come up with every possible combination of stimulus by hand
- Constrained random simulation is a staple of verification
  - Generation of random instruction streams controlled through a set of adjustable constraints
  - Great at hitting many common and uncommon design corners

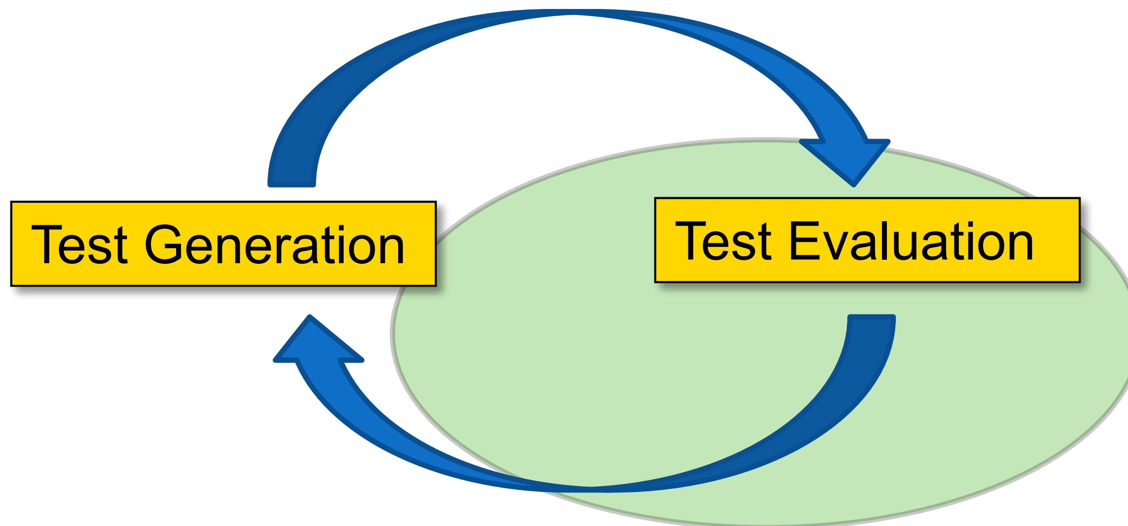
Background

- However, random testing is also inefficient and expensive!
- Random distributions hit most common cases most often, spending majority of the time testing the same things over and over
  - *Hard to find bugs* take a long time to find!



Two Ideas Presented

- A new type of *coverage*
  - A way to extract information about a single test, to provide feedback on its quality
- A way to use this feedback in machine learning algorithms
  - Optimization designed to find *hard to find bugs* quicker

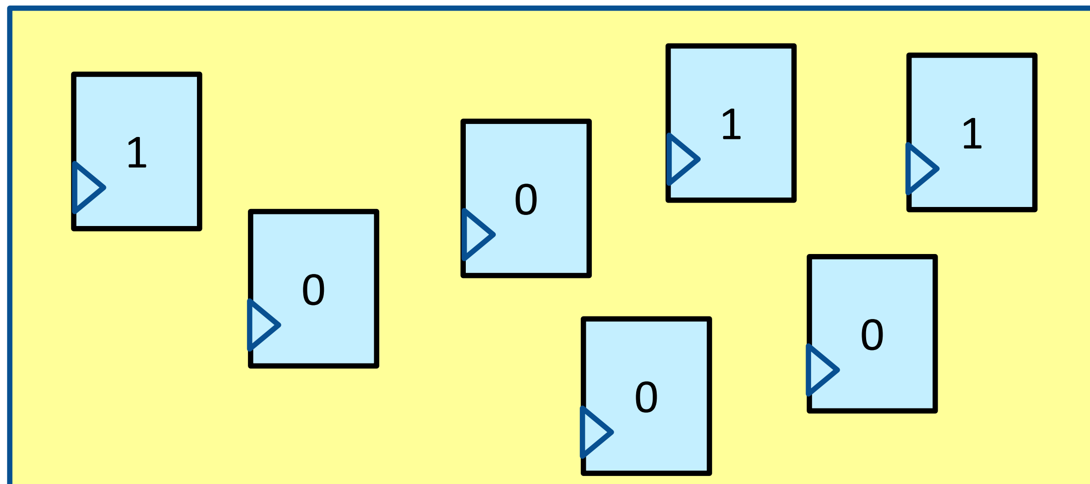


Finding hard-to-find bugs

- Non-trivial bugs require a combination of events and state changes to occur in close proximity
- Most bugs aren't particularly "deep"
  - It takes a couple of things to line up that we usually haven't thought to line up
- Verification engineers bias stimulus towards areas that are likely to cause bugs
  - Great use of experience and knowledge to find most bugs
  - However, we can't just keep running the same things
- Need an objective way to evaluate test variety and coverage
  - *Objective* is the key – we must eliminate the bias from hand-written functional coverage to find the *hard-to-find* corner cases

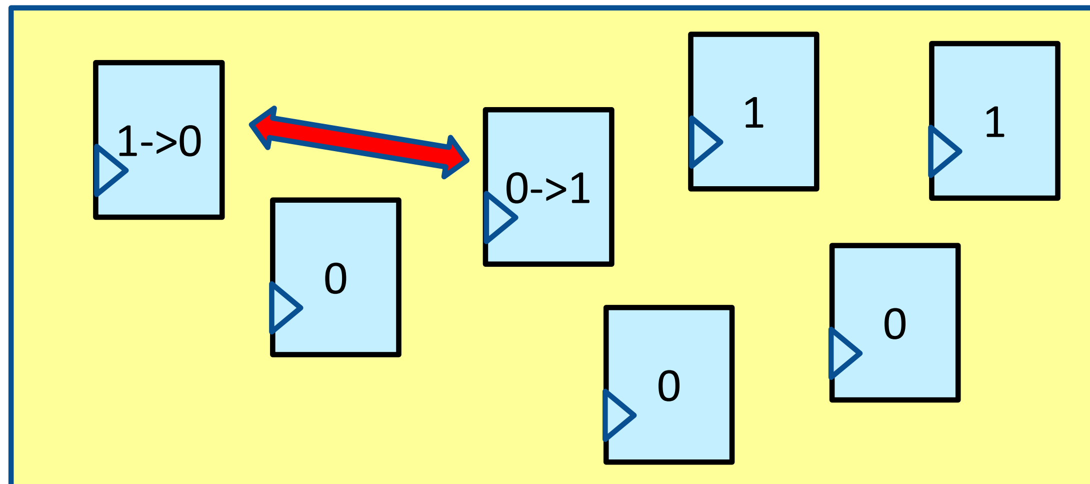
Exploring the state space

- One objective view of design coverage is its state space
  - State space of the design is represented by all of its flops
  - The total space size is 2^{flops}, which is not practical to track
- The interesting things happen when state changes
  - Flop toggle coverage – good start, but too simple, like CCOV



Lining things up

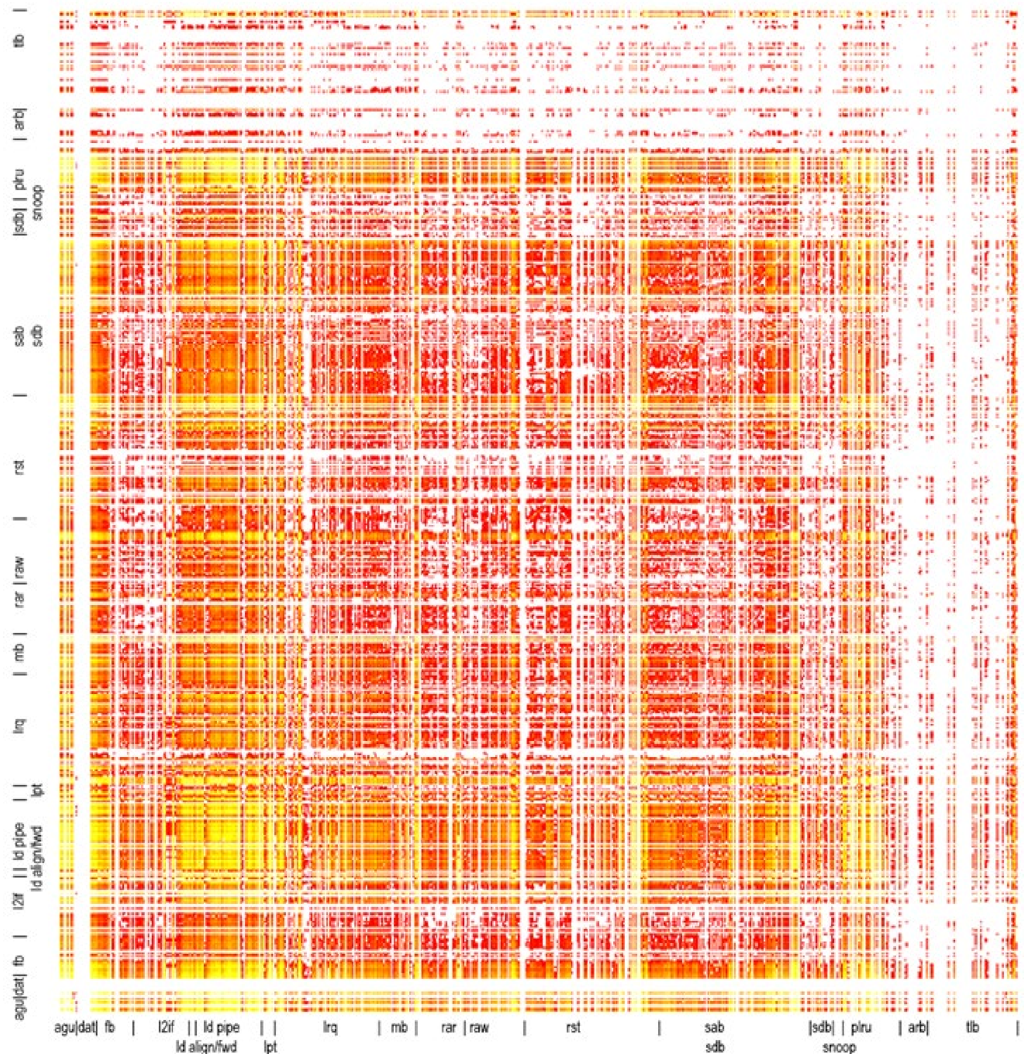
- Approximation for "events lining up" that takes design state into account:
  - Two flops toggling in close proximity in time
- Still fairly simple to track (state space is $flops^2$), but much more interesting than single flop toggle
- Very objective – requires no understanding of the design



2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Toggle matrix

- Yellow represent areas of high toggle counts, red are low, and white are blank
- Logarithmic scale – yellows are an order of magnitude higher than reds
- This represents one randomly picked test



Interpreting the results

- How many total *toggle pairs* a test produces:
  - indication of the *volume* of activity

- How many *toggle pairs* (bins) are exercised by the test:
  - indication of the *breadth* of the test

- We also need to focus on *hard to hit* bins that are rarely exercised
  - Don't bother optimizing for bins that are hit all the time
  - Filter anything that is easy to hit – bins hit by more than 50% of the tests is a good start

Scoring a Test

- Having a “score” for a test good for learning algorithms
- High score means:
  - High activity of rare events in the test (volume)
  - Many different rare events hit (breadth)
- Then, we calculate the score:

$$Score = (FilteredVolume^2 + Rare_Factor * FilteredBreadth^2)^{Power_Factor}$$

- Rare_Factor / Power_Factor provide easy tuning

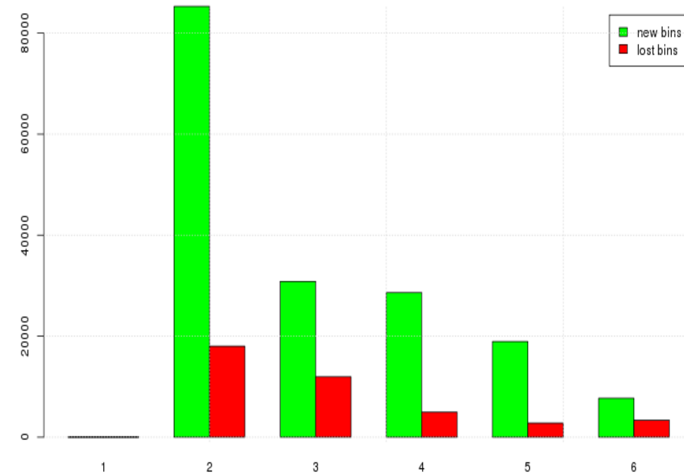
Machine Learning through a Genetic Algorithm

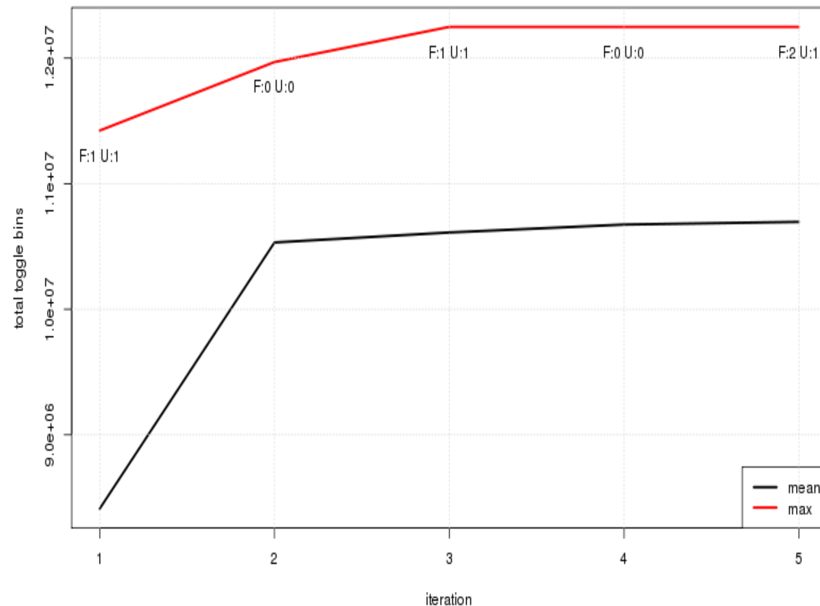
- A type of reinforced learning algorithm
  - Select a random *population* of tests, and evaluate each
  - Create the next *generation* of tests by:
    - *mutating* (slightly adjusting constraints) current tests
    - *mating* (take an average of two tests) current tests
  - The evaluation score dictates the chance of a test participating in the next generation

- Toggle pair coverage score used to select tests

Iteration Performance

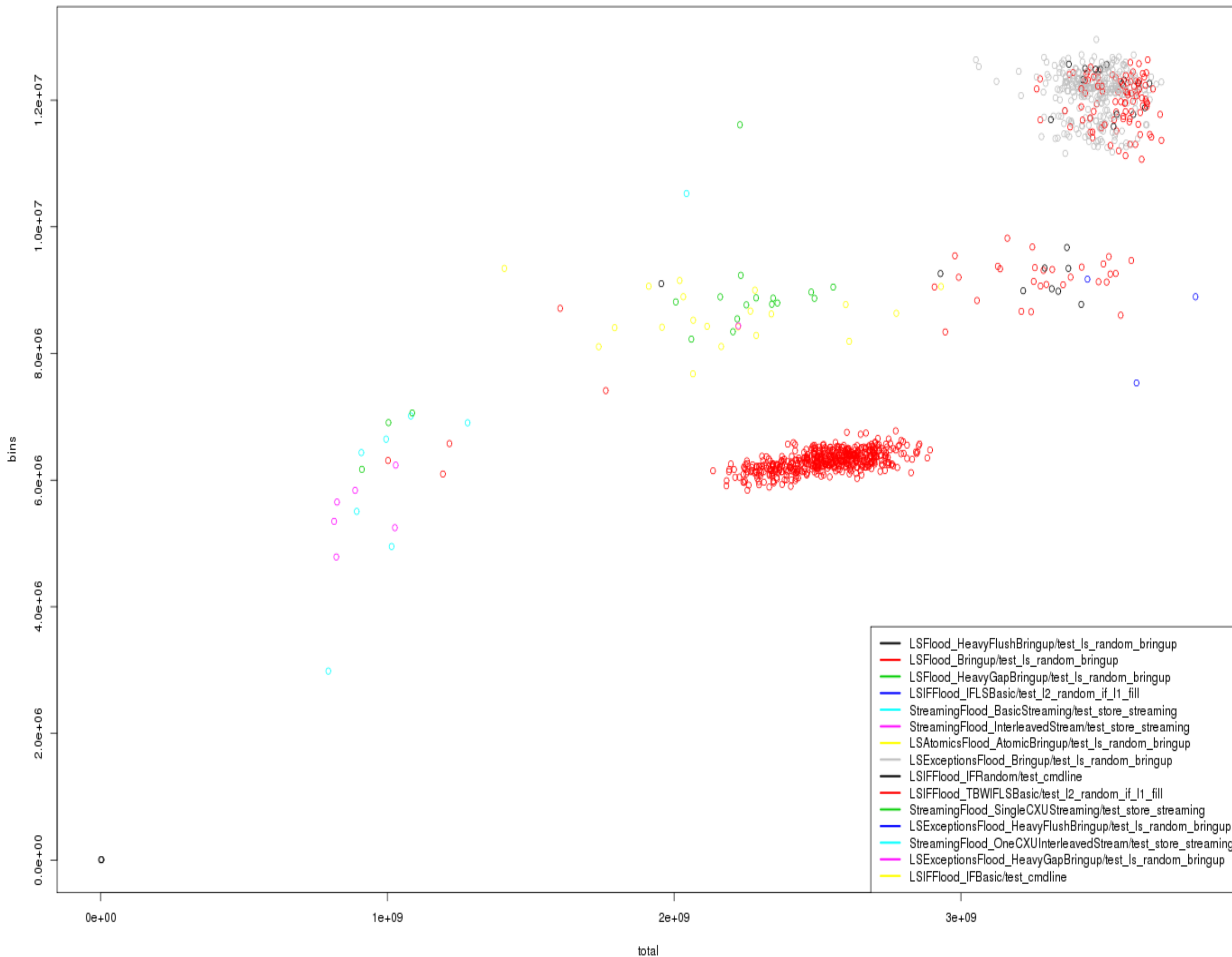
- Progress is charted through each iteration
- The iterations of interest are the ones that:
  - Show spikes over previous iterations
  - Show overall highest averages or totals
  - Have exposed new fail signatures





- It's important to monitor number of new bins hit, as well as bins "lost", i.e. bins that we no longer hit in the latest iteration (see above)

Volume vs Breadth over iterations



Does it find bugs?

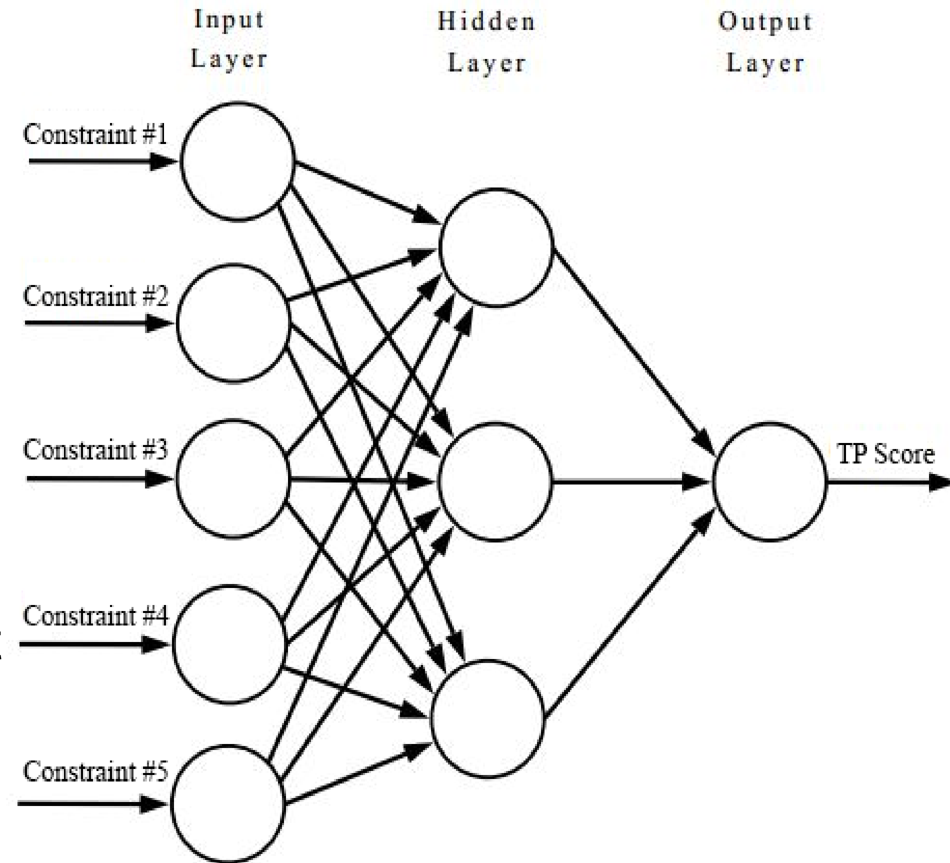
- Yes! It's still early, but the data is promising on LSU and L2
  - One of the iterations found a new bug, optimized large run found 3 more and failed over 450 times

Regression	Test Count	Fail Count	Pass Rate	Cycles	Unique Signatures
Regular weekly run	30000	24	99.92	173.6 Million	4
6 iterations of 500 tests	2749	41	98.5	15.4 Million	5
Large run using 6th iteration test selection	30000	469	98.43	166.1 Million	8

DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES 2017

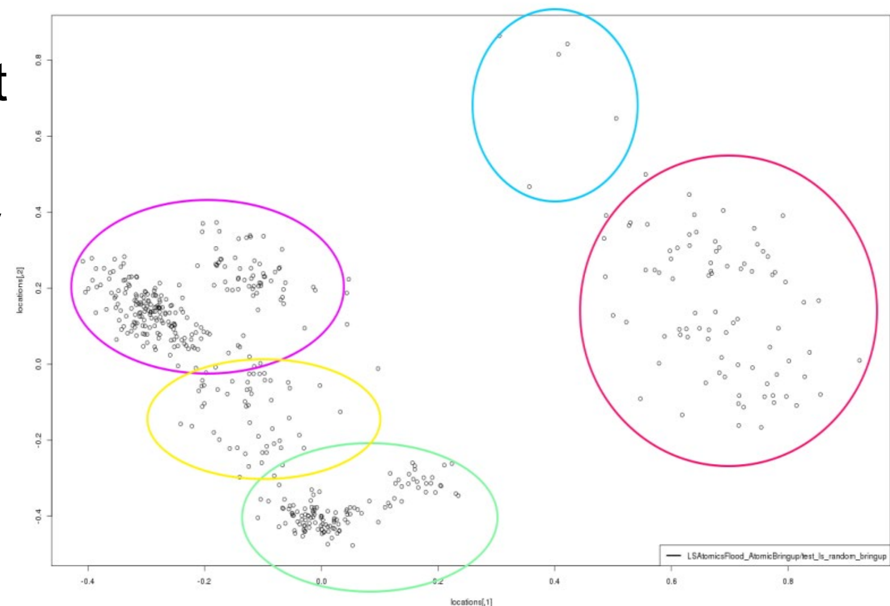
Other ML algorithms – NNs and SVMs

- Genetic algorithms require feedback on each test, making iterations slow
- If a neural network could be trained to predict a score for a test with reasonable accuracy, large sets of good tests could be generated much quicker
- Noisy results (due to random nature of tests) makes it difficult to train a network
  - Large amount of data needed
  - Filtering, principal component analysis



Other ML algorithms – Unsupervised Learning

- A clustering algorithm can detect groups of test that are “similar”
  - This can be used to “spread” the tests around
  - Run separate optimization on each cluster
- Anomaly detection
  - Algorithm that detects tests that the rest
  - This kind of a test is more likely corner cases



Next Steps

- This work is in early stages, and there are many ideas and trials to go through!

- Try other projects and designs
- Use meta-learning to learn the best GA parameters
- Continue to experiment with other ML algortihms

Questions?